REPRESENTING LATENT DATA IN AN EXTENSIBLE MARKUP LANGUAGE DOCUMENT

Copyright Notice

Field of the Invention

The present invention relates generally to utilization of styles and formatting settings in a computer-generated document. More particularly, the present invention relates to representing latent styles and formatting data in an Extensible Markup Language (XML) document.

Background of the Invention

Computer software applications allow users to create a variety of documents to assist them in work, education, and leisure. For example, popular word processing applications allow users to create letters, articles, books, memoranda, and the like. Spreadsheet applications allow users to store, manipulate, print, and display a variety of alphanumeric data. Such applications have a number of well-known strengths including rich editing, formatting, printing, calculation, and on-line and off-line editing.

A typical software application, such as a word processing application, may have tens or even hundreds of style and formatting properties that may be applied to text and/or data. For example, a word processing application may support many combinations of styles and formatting that may be selected by a user. For example, a style called "Header 1" may be selected by a user for automatically applying a certain font size, display and print justification, tab settings and the like. In addition to numerous settings that may be provided by application developers, many applications also allow a user to develop custom style and formatting settings.

According to prior systems, all of the possible style and formatting settings available to the user, whether used or not, are instantiated when the user saves the document. Unfortunately, when the user opens the document with a newer or different version of the software application that has the same or similar style settings but with different individual properties for the same or similar style settings, the user is stuck with the style or formatting settings instantiated for the document using the previous or creating application. Additionally, the software user may be allowed by his application to dictate that certain styles or formatting settings must be used or that certain styles or formatting settings may not be used with a particular document. In such cases, these styles or formatting settings are likewise instantiated upon document save which prevents the use of future changes to the styles or formatting settings allowed or disallowed by the user. Moreover, instantiation of all styles and formatting settings from a given software application causes the file size of a saved document to be excessive.

It is with respect to these and other considerations that the present invention has been made.

Summary of the Invention

Embodiments of the present invention solve the above and other problems by allowing styles and other formatting settings to remain latent until one or more particular styles or formatting settings is instantiated by a user. According to

aspects of the invention, all available styles and formatting properties are represented as individual latent objects. Data representing each latent style or formatting object is persisted in a data structure apart from the user's document. Each potential style setting or formatting setting may be set as locked for use for a given document or locked for non-use for a given document. Exceptions to locked for use or locked for non-use may be established by the user. Upon saving the document, no data is saved for the numerous available style and formatting settings in the document. Each available document style or formatting setting remains latent and is represented by a persisted data object. When a particular style is then called upon by a subsequent consuming application, that application may then instantiate the particular style or formatting setting based on the properties of that style or formatting setting available from the subsequent consuming application.

These and other features, advantages, and aspects of the present invention may be more clearly understood and appreciated from a review of the following detailed description of the disclosed embodiments and by reference to the appended drawings and claims.

Brief Description of the Drawings

Fig. 1 is a simplified block diagram of a computing system and associated peripherals and network devices that provide an exemplary operating environment for the present invention.

Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model.

Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality model.

Fig. 4 illustrates a computer-generated screen shot for allowing a user to selectively lock or unlock available style settings for application to a document.

Fig. 5 illustrates a computer-generated screen shot showing a document formatted according to a number of different style settings.

Fig. 6 illustrates a computer-generated screen shot showing a sample XML file according to embodiments of the present invention.

Detailed Description of the Preferred Embodiment

Embodiments of the present invention are directed to methods and systems for representing latent style and formatting settings data in an Extensible Markup Language (XML) document. In the following detailed description, references are made to the accompanying drawings that forma part hereof, and in which are shown by way of illustrations specific embodiments or examples. These embodiments may be combined, other embodiments may be utilized, and structural changes may be made without departing from the spirit or scope of the present invention. The following detailed description is therefore not to be taken in a limiting senses and the scope of the present invention is defined by the appended claims and their equivalents.

Referring now to the drawings, in which like numerals represent like elements through the several figures, aspects of the present invention and the exemplary operating environment will be described. Fig. 1 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which the invention may be implemented. While the invention will be described in the general context of program modules that execute in conjunction with an application program that runs on an operating system on a personal computer, those skilled in the art will recognize that the invention may also be implemented in combination with other program modules.

Generally, program modules include routines, programs, components, data structures, and other types of structures that perform particular tasks or implement particular abstract data types. Moreover, those skilled in the art will appreciate that the invention may be practiced with other computer system configurations, including hand-held devices, multiprocessor systems, microprocessor-based or programmable consumer electronics, minicomputers, mainframe computers, and the like. The invention may also be practiced in distributed computing environments where tasks are

performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote memory storage devices.

Turning now to Fig. 1, illustrative computer architecture for a personal computer 2 for practicing the various embodiments of the invention will be described. The computer architecture shown in Fig. 1 illustrates a conventional personal computer, including a central processing unit 4 ("CPU"), a system memory 6, including a random access memory 8 ("RAM") and a read-only memory ("ROM") 10, and a system bus 12 that couples the memory to the CPU 4. A basic input/output system containing the basic routines that help to transfer information between elements within the computer, such as during startup, is stored in the ROM 10. The personal computer 2 further includes a mass storage device 14 for storing an operating system 16, application programs, such as the application program 305, and data.

The mass storage device 14 is connected to the CPU 4 through a mass storage controller (not shown) connected to the bus 12. The mass storage device 14 and its associated computer-readable media, provide non-volatile storage for the personal computer 2. Although the description of computer-readable media contained herein refers to a mass storage device, such as a hard disk or CD-ROM drive, it should be appreciated by those skilled in the art that computer-readable media can be any available media that can be accessed by the personal computer 2.

By way of example, and not limitation, computer-readable media may comprise computer storage media and communication media. Computer storage media includes volatile and non-volatile, removable and non-removable media implemented in any method or technology for storage of information such as computer-readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EPROM, EEPROM, flash memory or other solid state memory technology, CD-ROM, DVD, or other optical storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by the computer.

According to various embodiments of the invention, the personal computer 2 may operate in a networked environment using logical connections to remote computers through a TCP/IP network 18, such as the Internet. The personal computer 2 may connect to the TCP/IP network 18 through a network interface unit 20 connected to the bus 12. It should be appreciated that the network interface unit 20 may also be utilized to connect to other types of networks and remote computer systems. The personal computer 2 may also include an input/output controller 22 for receiving and processing input from a number of devices, including a keyboard or mouse (not shown). Similarly, an input/output controller 22 may provide output to a display screen, a printer, or other type of output device.

As mentioned briefly above, a number of program modules and data files may be stored in the mass storage device 14 and RAM 8 of the personal computer 2, including an operating system 16 suitable for controlling the operation of a networked personal computer, such as the WINDOWS XP operating system from MICROSOFT CORPORATION of Redmond, Washington. The mass storage device 14 and RAM 8 may also store one or more application programs. In particular, the mass storage device 14 and RAM 8 may store an application program 305 for creating and editing an electronic document 310. For instance, the application program 305 may comprise a word processing application program, a spreadsheet application, a contact application, and the like. Application programs for creating and editing other types of electronic documents may also be used with the various embodiments of the present invention. A schema file 330 and a namespace/schema library 400, described below, are also shown.

Exemplary embodiments of the present invention are implemented by communications between different software objects in an object-oriented programming environment. For purposes of the following description of embodiments of the present invention, it is useful to briefly to describe components of an object-oriented programming environment. Fig. 2 is a simplified block diagram illustrating interaction between software objects according to an object-oriented programming model. According to an object-oriented programming environment, a first object 210 may include software code, executable methods, properties, and parameters. Similarly, a

second object 220 may also include software code, executable methods, properties, and parameters.

A first object 210 may communicate with a second object 220 to obtain information or functionality from the second object 220 by calling the second object 220 via a message call 230. As is well know to those skilled in the art of object-oriented programming environment, the first object 210 may communicate with the second object 220 via application programming interfaces (API) that allow two disparate software objects 210, 220 to communicate with each other in order to obtain information and functionality from each other. For example, if the first object 210 requires the functionality provided by a method contained in the second object 220, the first object 210 may pass a message call 230 to the second object 220 in which the first object identifies the required method and in which the first object passes any required parameters to the second object required by the second object for operating the identified method. Once the second object 220 receives the call from the first object, the second object executes the called method based on the provided parameters and sends a return message 250 containing a value obtained from the executed method back to the first object 210.

For example, in terms of embodiments of the present invention, and as will be described below, a first object 210 may be a third party customized application that passes a message to a second object such as an Extensible Markup Language schema validation object whereby the first object identifies a method requiring the validation of a specified XML element in a document where the specified XML element is a parameter passed by the first object with the identified method. Upon receipt of the call from the first object, according to this example, the schema validation object executes the identified method on the specified XML element and returns a message to the first object in the form of a result or value associated with the validated XML element. Operation of object-oriented programming environments, as briefly described above, are well known to those skilled in the art.

As described below, embodiments of the present invention are implemented through the interaction of software objects in the use, customization, and

application of components of the Extensible Markup Language (XML). Fig. 3 is a block diagram illustrating interaction between a document, an attached schema file, and a schema validation functionality module. As is well known to those skilled in the art, the Extensible Markup Language (XML) provides a method of describing text and data in a document by allowing a user to create tag names that are applied to text or data in a document that in turn define the text or data to which associated tags are applied. For example referring to Fig. 3, the document 310 created with the application 305 contains text that has been marked up with XML tags 315, 320, 325. For example, the text "Greetings" is annotated with the XML tag <title>. The text "My name is Sarah" is annotated with the <body> tag. According to XML, the creator of the <title> and <body> tags is free to create her own tags for describing the tags to which those tags will be applied. Then, so long as any downstream consuming application or computing machine is provided instructions as to the definition of the tags applied to the text, that application or computing machine may utilize the data in accordance with the tags. For example, if a downstream application has been programmed to extract text defined as titles of articles or publications processed by that application, the application may parse the document 310 and extract the text "Greetings," as illustrated in Fig. 3 because that text is annotated with the tag <title>. The creator of the particular XML tag naming for the document 310, illustrated in Fig. 3, provides useful description for text or data contained in the document 310 that may be utilized by third parties so long as those third parties are provided with the definitions associated with tags applied to the text or data.

According to embodiments of the present invention, the text and XML markup entered into the document 310 may be saved according to a variety of different file formats and according to the native programming language of the application 305 with which the document 310 is created. For example, the text and XML markup may be saved according to a word processing application, a spreadsheet application, and the like. Alternatively, the text and XML markup entered into the document 310 may be saved as an XML format whereby the text or data, any applied XML markup, and any formatting such as font, style, paragraph structure, etc. may be saved as an XML

representation. Accordingly, downstream or third party applications capable of understanding data saved as XML may open and consume the text or data thus saved as an XML representation. For a detailed discussion of saving text and XML markup and associated formatting and other attributes of a document 310 as XML, see U.S. Patent Application entitled "Word Processing Document Stored in a Single XML File that may be Manipulated by Applications that Understanding XML," U.S. Serial No. 10/187,060, filed June 28, 2002, which is incorporated herein by reference as if fully set out herein. An exemplary schema in accordance with the present invention is disclosed beginning on page 11 in an application entitled "Mixed Content Flexibility," Serial No. _____, Docket No. 60001.0275US01, filed December 2, 2003, which is hereby incorporated by reference in its entirety.

In order to provide a definitional framework for XML markup elements (tags) applied to text or data, as illustrated in Fig. 3, XML schema files are created which contain information necessary for allowing users and consumers of marked up and stored data to understand the XML tagging definitions designed by the creator of the document. Each schema file also referred to in the art as a Namespace or XSD file preferably includes a listing of all XML elements (tags) that may be applied to a document according to a given schema file. For example, a schema file 330, illustrated in Fig. 3, may be a schema file containing definitions of certain XML elements that may be applied to a document 310 including attributes of XML elements or limitations and/or rules associated with text or data that may be annotated with XML elements according to the schema file. For example, referring to the schema file 330 illustrated in Fig. 3, the schema file is identified by the Namespace "intro" the schema file includes a root element of <intro card>.

According to the schema file 330, the <intro card> element serves as a root element for the schema file and also as a parent element to two child elements <title> and <body>. As is well known to those skilled in the art, a number of parent elements may be defined under a single root element, and a number of child elements may be defined under each parent element. Typically, however, a given schema file 330 contains only one root element. Referring still to Fig. 3, the schema file 330 also

contains attributes 340 and 345 to the <title> and <body> elements, respectfully. The attributes 340 and 345 may provide further definition or rules associated with applying the respective elements to text or data in the document 310. For example, the attribute 345 defines that text annotated with the <title> element must be less than or equal to twenty-five characters in length. Accordingly, if text exceeding twenty-five characters in length is annotated with the <title> element or tag, the attempted annotation of that text will be invalid according to the definitions contained in the schema file 330.

By applying such definitions or rules as attributes to XML elements, the creator of the schema may dictate the structure of data contained in a document associated with a given schema file. For example, if the creator of a schema file 330 for defining XML markup applied to a resume document desires that the experience section of the resume document contain no more than four present or previous job entries, the creator of the schema file 330 may define an attribute of an <experience> element, for example, to allow that no more than four present or past job entries may be entered between the <experience> tags in order for the experience text to be valid according to the schema file 330. As is well known to those skilled in the art, the schema file 330 may be attached to or otherwise associated with a given document 310 for application of allowable XML markup defined in the attached schema file to the document 310. According to one embodiment, the document 310 marked up with XML elements of the attached or associated schema file 330 may point to the attached or associated schema file by pointing to a uniform resource identifier (URI) associated with a Namespace identifying the attached or associated schema file 330.

According to embodiments of the present invention, a document 310 may have a plurality of attached schema files. That is, a creator of the document 310 may associate or attach more than one schema file 330 to the document 310 in order to provide a framework for the annotation of XML markup from more than one schema file. For example, a document 310 may contain text or data associated with financial data. A creator of the document 310 may wish to associate XML schema files 330 containing XML markup and definitions associated with multiple financial institutions. Accordingly, the creator of the document 310 may associate an XML schema file 330

from one or more financial institutions with the document 310. Likewise, a given XML schema file 330 may be associated with a particular document structure such as a template for placing financial data into a desirable format.

According to embodiments of the present invention, a collection of XML schema files and associated document solutions may be maintained in a Namespace or schema library located separately from the document 310. The document 310 may in turn contain pointers to URIs in the Namespace or schema library associated with the one or more schema files attached to otherwise associated with the document 310. As the document 310 requires information from one or more associated schema files, the document 310 points to the Namespace or schema library to obtain the required schema definitions. For a detailed description of the use of an operation of Namespace or schema libraries, see U.S. Patent Application entitled "System and Method for Providing Namespace Related Information," U.S. Serial No. 10/184,190, filed June 27, 2002, and U.S. Patent Application entitled "System and Method for Obtaining and Using Namespace Related Information for Opening XML Documents," U.S. Serial No. 10/185,940, filed June 27, 2002, both U.S. patent applications of which are incorporated herein by reference as if fully set out herein. For a detailed description of a mechanism for downloading software components such as XML schema files and associated solutions from a Namespace or schema library, see US Patent Application entitled Mechanism for Downloading Software Components from a Remote Source for Use by a Local Software Application, US Serial No. 10/164,260, filed June 5, 2002.

Referring still to Fig. 3, a schema validation functionality module 350 is illustrated for validating XML markup applied to a document 310 against an XML schema file 330 attached to or otherwise associated with the document 310, as described above. As described above, the schema file 330 sets out acceptable XML elements and associated attributes and defines rules for the valid annotation of the document 310 with XML markup from an associated schema file 330. For example, as shown in the schema file 330, two child elements <title> and <body> are defined under the root or parent element <intro card>. Attributes 340, 345 defining the acceptable string length of text associated with the child elements <title> and <body> are also illustrated. As

described above, if a user attempts to annotate the document 310 with XML markup from a schema file 330 attached to or associated with the document in violation of the XML markup definitions contained in the schema file 330, an invalidity or error state will be presented. For example, if the user attempts to enter a title string exceeding twenty-five characters, that text entry will violate the maximum character length attribute of the <title> element of the schema file 330. In order to validate XML markup applied to a document 310, against an associated schema file 330, a schema validation module 350 is utilized. As should be understood by those skilled in the art, the schema validation module 350 is a software module including computer executable instructions sufficient for comparing XML markup and associated text entered in to a document 310 against an associated or attached XML schema file 330 as the XML markup and associated text is entered in to the document 310.

According to embodiments of the present invention, the schema validation module 350 compares each XML markup element and associated text or data applied to the document 310 against the attached or associated schema file 330 to determine whether each element and associated text or data complies with the rules and definitions set out by the attached schema file 330. For example, if a user attempts to enter a character string exceeding twenty-five characters annotated by the <title> elements 320, the schema validation module will compare that text string against the text string attribute 340 of the attached schema file 330 and determine that the text string entered by the user exceeds the maximum allowable text string length. Accordingly, an error message or dialogue will be presented to the user to alert the user that the text string being entered by the user exceeds the maximum allowable character length according to the attached schema file 330. Likewise, if the user attempts to add an XML markup element between the <title> and the <body> elements, the schema validation module 350 will determine that the XML markup element applied by the user is not a valid element allowed between the <title> and <body> elements according to the attached schema file 330. Accordingly, the schema validation module 350 will generate an error message or dialogue to the user to alert the user of the invalid XML markup.

Representing Latent Data in an Extensible Markup Language Document

As briefly described above, embodiments of the present invention allow style and other formatting settings to remain as latent data objects available for use with a computer-generated document across varying versions of a software application. Style and formatting properties available for use with a computer-generated document are represented in an Extensible Markup Language document as latent objects and are maintained in a separate data array so that each of the different styles and formatting properties are not instantiated upon document saving. Because the various style and formatting properties remain latent as represented in a separate data array, a future consuming application may instantiate only those style and formatting settings as required for a given document.

Fig. 4 illustrates a computer-generated screen shot for allowing a user to selectively lock or unlock available style settings for application to a document. As is appreciated by those skilled in the art, a typical software application, such as a word processing application, may have tens or even hundreds of different style and formatting settings available for application to text and data. A typical style setting might include a style setting called "header 1" in which a particular print size, font, paragraph structure and the like are established. Selection of this style and application to a portion of text or to an entire document will apply the properties of the style setting to the selected text or document. Because each style setting may include numerous properties and because a given software application may include tens or even hundreds of style settings, if each style setting is fully instantiated at the time of document save, file sizes become enormous. Moreover, instantiation of all available style settings may prevent the user of a future or varying version of the software application from utilizing different style settings other than those instantiated by the user upon document save.

Referring to Fig. 4, an exemplary user interface for allowing a user to select style settings that may be used with a particular document or text selection or for allowing the user to select style settings that may not be used for a document or given text selection. According to embodiments of the present invention, whether formatting

styles are marked as locked for use or locked for non-use, those style settings are not instantiated upon document save, but data representing which style settings are locked and which style settings are unlocked is maintained in a separate data array for future reference by a future application program opening a given document. For a detailed description of locking and unlocking styles and formatting changes for use in association with a computer-generated document, see United States Patent Application Serial No. 10/664,734, filed September 18, 2003, entitled "Method and Apparatus For Restricting The Application Of Formatting To The Contents Of An Electronic Document," applicant matter number 60001.0274US01/304205.1, which is expressly incorporated herein by reference as if fully set out herein.

Fig. 5 illustrates a computer-generated screen shot showing a document formatted according to a number of different style settings. As shown in Fig. 5, a document is illustrated whereby four separate style settings have been applied to text selections in the document.

Fig. 6 illustrates a computer-generated screen shot showing a sample XML file according to embodiments of the present invention. According to embodiments of the present invention, all style settings available for use with a given document are enumerated and data representing each style setting is saved as a latent data object in a separate data array apart from the document. For example, if 156 different style and/or formatting settings are available to the document according to the application program creating the document, a latent count of 156 is set. After the total number of latent style or formatting settings is enumerated, a default state of locked for use or locked for non-use is set for the entire set of enumerated latent style and formatting objects. Any exceptions to the default state are then made. For example, if the user defines that only styles "heading 5, heading 6 and heading 7" may be used for a given document, then all available style and formatting settings will be set to a default of locked for non-use and the style settings "heading 5, heading 6 and heading 7" will be set as exceptions to the default state. According to one embodiment of the invention, the default state may be applied to all style and formatting settings comprising a majority of the enumerated style and formatting settings while the remaining settings

will be set as exceptions. That is, if 100 file and formatting settings are available and 60 formatting settings are locked for non-use, then the default state will be set as locked for non-use and the remaining 40 style settings will be designated as exceptions to the default state.

Referring to Fig. 6, an XML representation of a word processing document 610 is illustrated. For representing the latent style and formatting objects saved to a separate data array for referenced by the document 610, an XML tag of <w:latentStyles> is utilized. Within the latent style tag, a default state property is set to "on or off" and a latent style count is enumerated. From the example shown in Fig. 6, a latent style count of 156 indicates that 156 different style or formatting settings are available and that the default state for the 156 style settings is "off" meaning that the default state for each of the 156 possible style and formatting settings is that they may be used with the document 610. Following with the example illustrated in Fig. 6, the user has set styles "heading 5, heading 6 and heading 7" as locked for non-use with the document. Accordingly, by saving the XML structure as illustrated in Fig. 6, a subsequent consuming application that is capable of parsing the XML structure will understand that of the 156 potential style and formatting settings for use with the document, styles "heading 5, heading 6 and heading 7" are exceptions to the default state and may not be used with the document. Advantageously, no XML representation must be made of each and every potential style or formatting setting or of the locked or unlocked state for each and every style or formatting setting which would increase file size and processing time dramatically. Thus, the subsequent XML parsing application needs only to reference the separate data array in which is maintained data representative of each of the potential style and formatting settings.

For example, if a user prepares a document in the United States using a U.S./English language-based XML schema file, and the user designates the latent style locked or unlocked states and exceptions as illustrated in Fig. 6, then the user may send the document to a colleague operating her computer in the Far East using a multilingual or Far East language XML schema file and XML parsing applications for parsing the document provided by the user. The colleague utilizing the document according to Far

East language settings will be prevented from using style settings "heading 5, heading 6 and heading 7" because the first user locked those style settings from use, but the colleague in the Far East will be able to utilize other style settings and will be able to use Far East particular fonts and other formatting properties assigned to the usable style settings using the multilingual or Far East XML schema file associated with the colleague's application program because the set of potential style or formatting settings have not been previously instantiated according to the first user's XML schema file. That is, if the colleague in the Far East wishes to use a style setting such as "heading 8" which has not been designated as an exception to the style lock state by the user, the colleague in the Far East may select style "heading 8" and style "heading 8" will be applied to the document according to the font and property characteristics available for that style based on the colleague's application program version or language settings. On the other hand, without the functionality of the present invention, all style settings whether locked or unlocked for use would be instantiated upon document save by the first user and would be saved out as part of the document save. Accordingly, the user's colleague in the Far East would not be able to apply Far East font properties available via the colleague's version of the application program because the properties for each of the potential style and formatting settings will already be populated in the manner in which they were instantiated by the first user who created the document.

As described herein, methods and systems are provided for allowing Extensible Markup Language representation of latent styles and formatting data in an XML document. It will be apparent to those skilled in the art that various modifications or variations may be made in the present invention without departing from the scope or spirit of the invention. Other embodiments of the invention will be apparent to those skilled in the art from consideration of the specification and practice of the invention disclosed herein.